Spectrum-Based Runtime Anomaly Localisation in Service-Based Systems

Qiang He, Xiaoyuan Xie, Feifei Chen, Yanchun Wang, Rajesh Vasa, Yun Yang
School of Software and Electrical Engineering
Swinburne University of Technology
Melbourne, Australia
{qhe, xxie, feifeichen, yanchunwang, rvasa, yyang}@swin.edu.au

Hai Jin
Services Computing Technology and Systems Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China
hjin@hust.edu.cn

Abstract—Runtime anomalies occurring to service-based systems (SBSs) that are dynamically composed of services must be located and fixed in a timely manner in order to guarantee timely and successful delivery of outcomes in response to user requests. Monitoring all component services constantly is impractical due to excessive resource consumption. Inspecting all component services upon anomalies is time-consuming and thus is also impractical. In this work, we propose an approach that employs spectrum-based fault localisation techniques to locate runtime anomalies in SBSs. Large-scale experiments are conducted and experimental results are presented to demonstrate the effectiveness and efficiency of the proposed approach.

Keywords—Service-Based System; Web Service; Quality of Service; Anomaly Localisation

I. INTRODUCTION

The service-oriented computing paradigm offers an effective way to engineer software systems [1] that are composed of services locally or remotely accessed by an execution engine (e.g., a BPEL engine [2]). In such a service-based system (SBS), the component services jointly offer the functionality of the SBS and collectively fulfil its users’ quality requirements.

Built from loosely coupled component services offered by independent (and often distributed) providers, SBSs operate in environments where key characteristics of the component services, such as the Quality of Service (QoS) properties, tend to be volatile. At runtime, various anomalies may occur to the basic component (BCs) of the SBS, e.g., unexpected workload changes, errors in the component services and failures of data transmissions, and impact on the quality of the SBS, causing end-to-end QoS violations. In this context, how to manage the quality of an SBS by detecting and adapting to runtime anomalies has become an important research direction [1, 3].

Response time, among various QoS dimensions, is of particular significance in quality management for SBSs. Amazon found that every 100ms of latency cost them 1% in sales [4] and Google found an extra 500ms seconds in search page generation time dropped traffic by 20% [5]. The increase in the number of time-constrained applications in the cloud, e.g., interactive and multimedia SBSs, is also driving the needs for response time management for SBSs [6]. Furthermore, the management of response time is the basis for the management of other QoS dimensions. On one hand, effective response time management promises better management of other QoS dimensions because many applications exhibit trade-offs between their response times and other QoS dimensions [7]. A video encoding application, for example, can often produce higher quality video if it is given more time to encode the video frames. On the other hand, the management of other QoS dimensions is tightly coupled with response time management. During execution, an SBS may need to be adapted to fix runtime anomalies. The adaptation itself takes time, and as a result, contributes to delaying the execution of the SBS. Thus, timely detection of runtime anomalies is significant to effective quality management for SBSs.

A straightforward solution to timely detection of runtime anomalies is to constantly monitor all the BCs of an SBS. In response to a detected anomaly, adaptation actions can be taken before performance degradation becomes noticeable by the users. However, monitoring may incur excessive costs [3], making it impossible to constantly monitor the entire SBS, especially in large-scale scenarios where the number of BCs of the SBS and the number of SBSs are huge. To address this issue, we proposed CriMon in [8] for formulating cost-effective monitoring strategies that focuses on the BCs on the critical path. However, what if an anomaly occurs to an unmonitored BC? When that happens, the unmonitored BCs must be inspected to pinpoint the anomaly. Unfortunately, a comprehensive inspection of all unmonitored BCs is potentially expensive and sometimes impractical due to two potential costs, i.e., resource cost and system cost. First, invocations of services may be charged if those services are owned and hosted by different organisations [9]. Even if the invocations are free, the negative impact on the SBS caused by the inspection of its BCs might further degrade its quality [3]. For example, monitoring sometimes involves sniffing network traffic and retrieving the logs of services’ and SBSs’ behaviour. As demonstrated in [10], those operations can result in up to 70% performance overhead, slowing down the entire SBS as a result. We identified that a quality inspection can cause as much as 40% performance overhead on a Web service under certain circumstances [11]. These issues do not exclude inspection of BCs as a promising way of detecting anomalies. Unmonitored BCs still need to be inspected to locate runtime anomalies. But those that are more likely to
be causing the QoS violation must be prioritised in the inspection to achieve timely anomaly detection at minimal resource cost and system cost.

To attack this challenge, we propose an approach that employs spectrum-based fault localisation (SFL) techniques [12] to locate runtime anomalies in SBSs. The reason for our choice is that SFL is one of the most lightweight fault localisation techniques [13]. An SBS can be represented by multiple execution scenarios [8]. In this context, the idea of SFL is to diagnose the differences in system spectra for passed and failed execution scenarios. Passed execution scenarios are the execution scenarios where QoS violations occurred, whereas failed execution scenarios are the execution scenarios where no QoS violation occurred. This information is used to calculate a heuristic measure for each BC of the SBS that expresses the suspiciousness of the BC being responsible for the QoS violation. The BCs can then be sorted in decreasing order of suspiciousness. Finally, the results are used to pinpoint the anomaly or, if an exact pinpoint cannot be found, presented as guidance that can reduce the SBS administrator’s effort in searching for the anomaly.

The contributions of this paper are twofold:
- Firstly, a spectrum-based approach is proposed to pinpoint anomalies upon end-to-end QoS violations in SBSs. This approach does not require constant monitoring or comprehensive inspection of all BCs of an SBS.
- Secondly, extensive and comprehensive experiments are conducted to evaluate our approach using a published real-world Web service dataset, which contains over 2500 real-world Web services.

The rest of this paper is organised as follows: Section II analyses the requirements with a motivating example. Section III reviews related work. Section IV describes our anomaly localisation approach for. Section V presents the experimental results and Section VI concludes the paper.

II. MOTIVATING EXAMPLE

This section presents an example SBS, namely OnlineLive, to motivate this research. As depicted in Figure 1, this SBS offers an on-demand service to convert, subtitle and transmit live video streams. OnlineLive consists of 26 BCs. $N_1$, $N_2$, ..., $N_9$ represent the component services and $E_1$, $E_2$, ..., $E_8$ represent the data transmissions between the component services. In response to a user request, the execution process of OnlineLive is as follows:

Step 1: $N_1$ splits the live media stream selected by the user into separate video and audio streams.

Step 2: The video and audio streams are processed in parallel. Specifically,
- For normal users, $N_2$ encodes 360p video stream and $N_3$ embeds advertisements into the video stream. For ad-free premium users, $N_4$ encodes 1080p video stream.
- $N_5$ generates the subtitle by performing speech recognition on the audio stream. Then, based on the user’s preference or country/region, the subtitle is sent to either $N_6$ or $N_7$ to be translated into one of the two optional languages.

Step 3: $N_6$ produces a media stream by merging and synchronising the video stream, audio stream and translated subtitle.

Step 4: The media stream is transmitted to the user.

OnlineLive must process the media stream timely and continuously. Otherwise, the user will receive a jittering media stream. When an anomaly occurs to a BC and causes delay to OnlineLive, the faulty BC must be identified in time so that adaptation actions can be taken to fix the anomaly immediately, avoiding or reducing the delay caused to the system.

III. RELATED WORK

When operating in volatile environments, SBSs must be monitored in order to achieve timely and accurate detection and prediction of runtime anomalies. A great deal of monitoring techniques and approaches have been proposed for SBSs. Several languages are proposed specifically for monitoring SBSs, such as WSCoL [3] and SECMOL [14]. These languages can be used to specify monitoring rules, constraints and assertions.

Many frameworks have also been proposed for monitoring SBSs. To name a few, the monitoring assertions specified using Barbon et al. [15] propose Astro, a monitoring solution aiming at separating the business logic of a Web service from its monitoring functionality. Baresi et al. [16] propose a general and comprehensive solution for monitoring service compositions. In this monitoring solution, monitoring constraints can be defined on single and multiple instances, on punctual properties and on complete behaviours. Guinea et al. [17] propose a framework that integrates monitoring across the software and infrastructure layers. A variation of the MAPE control loop is introduced into the framework that acknowledges the multi-faceted nature of SBSs. Kertzés et al. [18] integrate SALMon [19] in IMA4SSP - their monitoring approach to seamless service provisioning - so that it can collect dynamic reliability information of SBSs expressed in pre-defined quality metrics.

Monitoring only the critical parts of an SBS is a cost-effective solution [20] to SBS monitoring. As a result, non-critical BCs will not be monitored. When an anomaly occurs to such a BC, it must be localised quickly so that adaptation actions can be taken to fix the anomaly immediately,
avoiding or reducing the delay caused to the system execution [1, 3, 21]. A great deal of fault localisation techniques have been proposed for different types of software systems. To name a few, Artzi et al. [22] presents how fault localisation algorithms can be enhanced to pinpoint faults effectively in web applications written in PHP. Park et al. [23] presents a dynamic fault localisation technique that can pinpoint faults in concurrent programs. Novotný et al. [24] presents a method for localising faults occurring in service-based systems on mobile ad hoc networks. Sharma et al. [25] presents a time-invariant relationships based approach for fault localisation in distributed systems. Nguyen et al. [26] presents FChain, a black-box online fault localisation approach for diagnosing performance anomalies in cloud systems. It assumes all the components of a cloud system are deployed within one cloud and thus all system metrics can be discovered immediately upon a detected performance anomaly. Thus, FChain is not suitable for SBSs who component services are very often distributed across multiple clouds.

An SBS is different from a traditional standalone software system as it usually consists of multiple distributed component services, whose status is difficult to inspect upon the occurrence of runtime anomalies. However, the execution engine of the SBS, e.g., the BPEL engine, is centralised. The end-to-end response time of the SBS can be collected easily and system delay can be detected efficiently. Taking this advantage, this paper applies the fault localisation technique to attack the challenging research problem of runtime anomaly localisation for SBSs.

IV. ANOMALY LOCALISATION

The most straight-forward solution to timely detection of runtime anomalies is to constantly monitor all the BCs of the SBS. Another solution is to comprehensively inspect the status and quality of all BCs every time a QoS violation occurs. As discussed in Section I, both solutions are expensive and sometimes impractical.

Another approach is to employ end-to-end QoS information of the SBS for localising the occurring anomaly. For example, four execution scenarios can be identified from OnlineLive (see Figure 1): es1={EP1, EP3}, es2={EP1, EP4}, es3={EP2, EP3} and es4={EP2, EP4}, as presented in Figure 2. Which one will be executed in response to a user request is dependent on the runtime decisions made at the two branch structures. Assume that the constraint for the response time of OnlineLive is 600ms but now it is taking OnlineLive 1000ms to process user requests, resulting in a QoS violation. System logs indicate that the current end-to-end response times of es1, es2, es3, and es4 are 800ms, 400ms, 1000ms and 600ms respectively. Obviously, es1 and es3 have violated the constraint of response time, thus there must be at least one faulty BC in both of these two execution scenarios. By comparing these four execution scenarios, we can find that E6, N6 and E8 are exclusively invoked by these two failed execution scenarios; E5, N7 and E9 are only involved in the passed execution scenarios; while the rest BCs are included in both cases. Therefore, we can reasonably infer that EL, N6 and EN are the most suspicious BCs that result in the violation of the time constraint.

Our anomaly localisation approach is designed as a five-phase process as shown in Figure 3. In Phase 1, the loops in the SBS are peeled. Then, in Phase 2, a set of execution scenarios are identified from the SBS. After that, in Phase 3, a coverage matrix is constructed. In Phase 4, similarity coefficients are calculated for the BCs of the SBS to indicate their suspiciousness of resulting in the anomaly. Finally, in Phase 5, the BCs are ranked to suggest an order in which the BCs should be inspected. Details of these phases are presented in Section IV.A to Section IV.E respectively.

Figure 2. Anomaly localisation procedure.
A. Phase 1: Loop Peeling

In this research, we use four types of basic compositional structures, i.e., sequence, branch, loop and parallel for representing and analyzing SBS. These compositional structures are included in BPMN [27], addressed by BPEL [2] - the de facto standards for specifying service-oriented business processes. They are also adopted in many other studies on SBS [8, 9, 28].

• Sequence. In a sequence structure, the BCs are executed one by one.

• Branch. In a branch structure, only one branch is selected for execution. For a set of branches \( \{b_1, ..., b_n\} \), the execution probability distribution \( \{p(b_1), ..., p(b_n)\} \), \( 0 \leq p(b) \leq 1, \sum_{i=1}^{n} p(b_i) = 1.0 \) is specified, where \( p(b_i) \), \( i=1, ..., n \), is the probability that the \( i^{th} \) branch is selected for execution.

• Loop. In a loop structure, the loop is executed for \( n \) \((0 \leq n \leq MNI)\) times. For a loop, the probability distribution \( \{p_0, ..., p_{MNI}\} \), \( 0 \leq p_i \leq 1, \sum_{i=0}^{MNI} p_i = 1.0 \) is specified, where \( p_i \), \( i=0, ..., MNI \), is the probability that the loop iterates for \( i \) times and \( MNI \) is the expected maximum number of iterations for the loop.

• Parallel. In a parallel structure, all the branches are executed at the same time.

\[ p(b_i), p_i, \text{ and } MNI \text{ can be evaluated based on the past executions of the SBS or can be specified by the developer [28]. We assume that for a loop, the } MNI \text{ can be determined or estimated. Otherwise, without an upper bound for the number of iterations, the execution times of the execution paths that contain the loop cannot be calculated since the loop may iterate infinitely.} \]

We represent service compositions using UML activity diagrams, where the nodes represent component services and the edges represent data transmissions. Without losing generality, we assume that a service composition is characterised by only one entry point and one exit point, and only includes structured loops with only one entry point and one exit point. If a service composition includes loops, we peel the loops by representing loop iterations as a set of branches with corresponding execution probabilities [28]. Figure 4 gives an example of peeling a loop structure \((MNI=2)\) by transforming it into a branch structure that contains three branches \( b_1, b_2 \) and \( b_3 \), where \( p_0, p_1, \text{ and } p_2 \) are the probabilities that \( b_1, b_2 \) and \( b_3 \) are selected for execution respectively. (Note that the first branch \( b_1 \) is selected if the loop iterates for \( 0 \) times, i.e., corresponding to \( p_0 \)).

B. Phase 2: Execution Scenario Identification

In a service composition where branches or loops are involved, different execution paths may be selected for execution. Thus, multiple execution scenarios can be identified from the service composition. These execution scenarios do not contain branch or loop structures, and hence can be modelled as Directed Acyclic Graphs (DAGs). As depicted in Figure 3, four execution scenarios can be identified from OnlineLive: \( es_1=\{EP_1, EP_3\}, es_2=\{EP_1, EP_4\} \), \( es_3=\{EP_2, EP_3\} \) and \( es_4=\{EP_2, EP_4\} \).

The identified execution scenarios will be used in the next section to construct the coverage matrix.

C. Phase 3: Coverage Matrix Construction

Anomaly localisation requires analysis of the differences in program spectra [29] for individual execution scenarios of the SBS. A program spectrum is a collection of execution trace that shows which parts (e.g., blocks and statements) of a program were included during an execution of the program. Many different forms of program spectra exist [30]. In this research, we propose the concept of BC-hit spectrum. A BC-hit spectrum contains a flag for every BC of an SBS, which indicates whether or not that BC is included in a particular execution scenario of the SBS. Take OnlineLive for example, there are four BC-hit spectra, one for each execution scenario, as presented by the four grey columns in Table I.

The BC-hit spectra of \( n \) execution scenarios constitute an \( n \times m \) binary coverage matrix, where \( m \) is the number of BCs in an SBS, as presented in Figure 5. Given \( i \in [1,...,n] \) and \( j \in [1,...,m] \), \( c_{ij}=1 \) indicates that the \( j^{th} \) BC is included in the \( i^{th} \) execution scenario. The four grey columns in Table 1 constitute the coverage matrix for OnlineLive.

The coverage matrix will be used in the similarity coefficient calculation for anomaly localisation discussed in Section IV.D.
which execution scenario each use runtime decisions made in the branch structures of the SBS managed by the execution engine that invokes the distributed services is with marginal overhead by calculating the similarity coefficient score is calculated based on a binary and the columns of the coverage matrix indicate the occurrence of the QoS violation vector needs to be logged. It can be obtained from the execution engine of the SBS with marginal overhead by calculating the time difference between the arrival of user requests and the delivery of corresponding outcomes. This is feasible because the execution engine that invokes the distributed services is managed by the SBS provider centrally [31]. Each request will traverse one execution scenario, depending on the runtime decisions made in the branch structures of the SBS. Which execution scenario each user request traverses also needs to be recorded to complete the QoS violation vector in Figure 5. This can be achieved with marginal overhead by recording the runtime decisions made at the branches by the execution engine of the SBS. Now assume that an anomaly is occurring to $N_a$, slowing it down and violating the constraint for the response time of OnlineLive. Because $N_a$ is included in $E_1$, the QoS violation will only be observed in $e_{S_1}$ and $e_{S_3}$. The QoS violation vector will be: $[1, 0, 1, 0]$. Table I presents the coverage matrix and QoS violation vector constructed for OnlineLive.

The similarity between the QoS violation vector of binary and the columns of the coverage matrix indicate the suspiciousness of the BCs being responsible for the occurring QoS violation. In Table I, only the rows corresponding to $N_a$, $E_i$ and $E_k$ are consistent with the QoS violation vector – they are all [1, 0, 1, 0]. Thus, $N_a$, $E_i$ and $E_k$ can be identified as the most suspicious BCs and should be inspected with the highest priority.

For large scale applications, the coverage matrices and the QoS violation vectors can be remarkably large and complex. The goal of anomaly localisation is to identify the BCs whose corresponding rows in the coverage matrix are most statistically similar to the QoS violation vector. The statistical similarity between each of the columns in the coverage matrix and the QoS violation vector can be quantified by calculating the similarity coefficients of the BCs. Calculated in different ways, many similarity coefficients exist, e.g., Jaccard coefficient $c_J$, used by the Pinpoint tool [32], the coefficient $c_T$ used in the Tarantula fault localisation tool [33] and the Ochiai coefficient $c_O$ often used in the molecular biology domain:

$$c_j(S_j) = \frac{n_{1j}(S_j)}{n_{11}(S_j) + n_{0j}(S_j) + n_{10}(S_j)}$$

\[D. \text{ Phase 4: Similarity Coefficient Calculation}\]

A similarity coefficient score is calculated based on a BC’s involvement in all the passed and failed execution scenarios. A high similarity coefficient score implies high suspiciousness.

In Figure 5, the binary QoS violation vector that contains $n$ flags is used to represent which execution scenarios are experiencing a delay in processing user requests. Given $i \in [1, \ldots, n]$, $v_i = 1$ indicates that $e_{S_i}$ is experiencing the delay, and 0 otherwise. To complete the QoS violation vector, the time consumed by each execution scenario in processing user requests needs to be logged. The similarity coefficient score is calculated based on a binary and the columns of the coverage matrix indicate the occurrence of the QoS violation vector needs to be logged. It can be obtained from the execution engine of the SBS with marginal overhead by calculating the time difference between the arrival of user requests and the delivery of corresponding outcomes. This is feasible because the execution engine that invokes the distributed services is managed by the SBS provider centrally [31]. Each request will traverse one execution scenario, depending on the runtime decisions made in the branch structures of the SBS. Which execution scenario each user request traverses also needs to be recorded to complete the QoS violation vector in Figure 5. This can be achieved with marginal overhead by recording the runtime decisions made at the branches by the execution engine of the SBS. Now assume that an anomaly is occurring to $N_a$, slowing it down and violating the constraint for the response time of OnlineLive. Because $N_a$ is included in $E_1$, the QoS violation will only be observed in $e_{S_1}$ and $e_{S_3}$. The QoS violation vector will be: $[1, 0, 1, 0]$. Table I presents the coverage matrix and QoS violation vector constructed for OnlineLive.

The similarity between the QoS violation vector of binary and the columns of the coverage matrix indicate the suspiciousness of the BCs being responsible for the occurring QoS violation. In Table I, only the rows corresponding to $N_a$, $E_i$ and $E_k$ are consistent with the QoS violation vector – they are all [1, 0, 1, 0]. Thus, $N_a$, $E_i$ and $E_k$ can be identified as the most suspicious BCs and should be inspected with the highest priority.

For large scale applications, the coverage matrices and the QoS violation vectors can be remarkably large and complex. The goal of anomaly localisation is to identify the BCs whose corresponding rows in the coverage matrix are most statistically similar to the QoS violation vector. The statistical similarity between each of the columns in the coverage matrix and the QoS violation vector can be quantified by calculating the similarity coefficients of the BCs. Calculated in different ways, many similarity coefficients exist, e.g., Jaccard coefficient $c_J$, used by the Pinpoint tool [32], the coefficient $c_T$ used in the Tarantula fault localisation tool [33] and the Ochiai coefficient $c_O$ often used in the molecular biology domain:

$$c_j(S_j) = \frac{n_{1j}(S_j)}{n_{11}(S_j) + n_{0j}(S_j) + n_{10}(S_j)}$$

\[9\]
process presented
picked
generate and introduce
During the execution of
the example SBS
is 3/21=1/7, or 14.29%.
In a worst case scenario, the cost of pinpointing the
ranking of
versa. For example, in Table 1 for Figure 1, the level
low localisation cost indicates high effectiveness and vice
by other r
E.
Experimental Setup
In the first series of experiments, we study the
effectiveness of our approach by comparing with random
inspection in response to runtime anomalies. To measure the
effectiveness of our approach, we use the localisation cost,
i.e., the percentage of BCs of an SBS that must be inspected
before reaching the faulty BC. This measure has been used
by other researchers [34, 35] for fault localisation studies. A
low localisation cost indicates high effectiveness and vice
versa. For example, in Table 1 for Figure 1, the level-2
ranking of \( N_0 \) (the faulty BC) is 3, same as \( E_L \) and \( E_X \). The
system administrator can choose to inspect all the suspicious
BCs at the same time or to inspect them at a random order.
In a worst case scenario, the cost of pinpointing the anomaly
is 3/21=1/7, or 14.29%. The evaluation process mimicked
the example SBS OnlineLive presented in Section II. During the execution of OnlineLive in each run, we
generate and introduce one runtime anomaly to a randomly
picked BC to simulate a volatile operating environment.
Upon anomalies, we follow the anomaly localisation process presented in Figure 2 to pinpoint the faulty BCs.

In the second series of experiments, we evaluate the
effectiveness of our approach in detecting multiple
anomalies. In each run of the experiment in this series, a
specific number of anomalies are introduced to multiple
randomly selected BCs of OnlineLive. Then, we follow the
procedure presented in Figure 2 and try to pinpoint all the
faulty BCs. For comparison purpose, we also employ the
random inspection approach by selecting random BCs for
inspection until all faulty BCs are pinpointed.

In the third series of experiments, we evaluate the
efficiency of our approach measured by the computational
overhead in calculating the coefficients and ranking the BCs,
i.e. phases 4 and 5 in the anomaly localisation procedure as
presented in Figure 2. The reason is that phases 1, 2 and 3
are performed offline at build-time. Phases 4 and 5 are
performed at runtime and directly contribute to the total
consumption for system recovery upon runtime anomalies. In this series of experiments, we utilise randomly
generated SBSs with different number of BCs. The number
of BCs increases from 100 to 1000 in steps of 100.

For each set of experiment, we average the results
obtained from 100 runs. All experiments are conducted on
a machine with Intel i5-4570 CPU 3.20GHz and 8 GB RAM,
running Windows 7 x64 Enterprise.

B. Experimental Results

Figure 6 compares the localisation cost of our
approaches with random inspection. Three similarity
coefficients are evaluated, i.e., \( c_f \) (see formula (9)), \( c_T \) (see
formula (10)) and \( c_O \) (see formula (11)). As we expected,
random inspection has to inspect approximately 50% of
the BCs to pinpoint the faulty BC. Our approaches by \( c_{f} \), \( c_{T} \) and
\( c_{O} \) incur an average localisation cost of 0.11, 0.19 and 0.11,
outperforming random inspection by a margin of 0.41, 0.33
and 0.41 respectively. \( c_{f} \) and \( c_{O} \) incur very similar
localisation cost across all cases, both holding an advantage
of 0.08 over \( c_{T} \). Take an SBS with 20 services for example.
Using our approach by \( c_{O} \), the system administrator needs to
inspect an average of 2 services to pinpoint the faulty
service. With the random approach, the system
administrator has to inspect an average of 10 services to
finally pinpoint the faulty service. Our approaches can

\[ c_T(S_j) = \begin{cases} 1.00 & \text{if } n_{o}(S_j) + n_{w}(S_j) = 0 \\ \frac{n_{e}(S_j)}{n_{o}(S_j) + n_{w}(S_j)} & \text{otherwise} \end{cases} \]

\[ c_O(S_j) = \sqrt{(n_{11}(S_j) + n_{01}(S_j)) \times (n_{11}(S_j) + n_{10}(S_j))} \]

where \( S_j \) is a BC, \( n_{o}(S_j) = |\{i | x_i = p \land v_i = q\}| \), \( p, q \in \{0, 1\} \).
\( x_i=1 \) indicates that \( S_j \) is included in \( e_s \), and 0 otherwise, \( v_i=1 \) indicates that QoS violation is occurring to \( e_s \) (i.e., a failed
execution scenario) and 0 (a passed execution scenario)
otherwise. For instance, \( n_{01}(S_j) \) and \( n_{11}(S_j) \) are the numbers
of passed and failed execution scenarios respectively where
\( S_j \) is involved. The accuracy of the three similarity
coefficients in the context of SBSs will be evaluated in the experiments.

E. Phase 5: Ranking

Based on the similarity coefficient scores, the BCs can
be ranked from high to low, where the highest indicates the
most unreliable. When multiple BCs share a same
coefficient score at one of the three levels, we use the
approach reported in [34]: all tied BCs get the greatest
ranking number. The Ranking column in Table I presents
the rankings of all BCs in OnlineLive.

V. EXPERIMENTS

We conduct two series of experiments in a simulated
volatile environment to evaluate our anomaly localisation
approach.

A. Experimental Setup

In the first series of experiments, we study the
effectiveness of our approach by comparing with random
inspection in response to runtime anomalies. To measure the
effectiveness of our approach, we use the localisation cost,
i.e., the percentage of BCs of an SBS that must be inspected
before reaching the faulty BC. This measure has been used
by other researchers [34, 35] for fault localisation studies. A
low localisation cost indicates high effectiveness and vice
versa. For example, in Table 1 for Figure 1, the level-2
ranking of \( N_0 \) (the faulty BC) is 3, same as \( E_L \) and \( E_X \). The
system administrator can choose to inspect all the suspicious
BCs at the same time or to inspect them at a random order.
In a worst case scenario, the cost of pinpointing the anomaly
is 3/21=1/7, or 14.29%. The evaluation process mimicked
the example SBS OnlineLive presented in Section II. During the execution of OnlineLive in each run, we
generate and introduce one runtime anomaly to a randomly
picked BC to simulate a volatile operating environment.
Upon anomalies, we follow the anomaly localisation process presented in Figure 2 to pinpoint the faulty BCs.

In the second series of experiments, we evaluate the
effectiveness of our approach in detecting multiple
anomalies. In each run of the experiment in this series, a
specific number of anomalies are introduced to multiple
randomly selected BCs of OnlineLive. Then, we follow the
procedure presented in Figure 2 and try to pinpoint all the
faulty BCs. For comparison purpose, we also employ the
random inspection approach by selecting random BCs for
inspection until all faulty BCs are pinpointed.

In the third series of experiments, we evaluate the
efficiency of our approach measured by the computational
overhead in calculating the coefficients and ranking the BCs,
i.e. phases 4 and 5 in the anomaly localisation procedure as
presented in Figure 2. The reason is that phases 1, 2 and 3
are performed offline at build-time. Phases 4 and 5 are
performed at runtime and directly contribute to the total
time consumption for system recovery upon runtime anomalies. In this series of experiments, we utilise randomly
generated SBSs with different number of BCs. The number
of BCs increases from 100 to 1000 in steps of 100.

For each set of experiment, we average the results
obtained from 100 runs. All experiments are conducted on
a machine with Intel i5-4570 CPU 3.20GHz and 8 GB RAM,
running Windows 7 x64 Enterprise.
significantly speed up the anomaly detection process. Adaptation actions can be taken immediately to fix the system, relieving or eliminating the QoS violation.

Figure 7 compares the localisation cost of our approaches with random inspection upon multiple anomalies.

Figure 8 shows the computational overheads of online similarity coefficient calculation.

VI. CONCLUSIONS

In this paper, we propose a spectrum-based approach for anomaly localisation in service-based systems. The main idea is to exploit end-to-end system QoS data for localising runtime anomalies. The program spectrum of the system execution scenarios are analysed to calculate the basic components’ suspiciousness of resulting in the runtime anomaly. The comprehensive experimental analysis shows the effectiveness and efficiency of our approach.

Our approach relies on the detection of SBS’s end-to-end response time violation. We assume that a runtime anomaly would always cause end-to-end response time violation. In most cases, this assumption is reasonable. However, a runtime anomaly occurring on a short end-to-end execution path might not lead to an end-to-end response time violation. Our approach cannot localise such a runtime anomaly. We will address this issue in our future work.

ACKNOWLEDGMENT

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

REFERENCES


